Scientific Visualization on Sparse Grids

Christian Teitzel
Computer Graphics Group
Am Weichselgarten 9
91058 Erlangen
Germany
teitzel@informatik.uni-erlangen.de

Matthias Hopf Thomas Ertl
Visualization and Interactive Systems Group
Breitwiesenstr. 20–22
70565 Stuttgart
Germany
{hopf,ertl}@informatik.uni-stuttgart.de

Abstract

The ever growing size of data sets resulting from industrial and scientific simulations and measurements have created an enormous need for analysis tools allowing interactive visualization. A promising hierarchical approach in the area of numerical simulation is called sparse grids. We present two major visualization algorithms working directly on the sparse grid representation of the data set. One of them is interactive particle tracing, which continues to be an important utility for evaluating CFD simulations. The other one is volume ray casting, which is of interest in many areas dealing with three-dimensional scalar data. Additionally we have been able to employ texture hardware support for the necessary function interpolation. Hence, we are able to perform volume visualization methods on compressed data sets at interactive frame rates, which is not possible with other methods like wavelets or fractal compression. In particular, we are able to handle sparse grids of level 13, which correspond to regular volumes of 8193^3 *voxels.*

1. Introduction

The sparse grid idea was developed in the 1960s by Babenko [1] and Smolyak [23]. They showed that a special tensor product technique of constructing higher dimensional quadrature formulas and approximation operators from corresponding one-dimensional objects leads to almost optimal error rates. In 1990 sparse grids were introduced to the field of numerical computation by Zenger [31]. By means of sparse grids, it is possible to reduce the total amount of data points or the number of unknowns in discrete partial differential equations. Due to these benefits, sparse grids are more and more used in numerical simulations [2, 3, 10, 11].

On the other hand, it is rather difficult to visualize the results of the simulation process directly on sparse grids, since evaluation and interpolation of function values is quite complicated. Because of this, the results of numerical simulations on sparse grids are usually interpolated to the associated full grid. Thereafter, all well known visualization algorithms working on regular grids can be used, e.g. particle tracing, iso-surface extraction, and volume rendering. However, a major drawback of this procedure is the fact that the advantage of low memory consumption of sparse grids comes to nothing using the associated full grid for the visualization step.

Therefore, visualization tools working directly on sparse grids are an important topic of research. Heußer and Rumpf presented an algorithm for iso-surface extraction on sparse grids [14]. In a previous work, we introduced particle tracing [26] and volume visualization [27] on uniform sparse grids.

The goal of this paper is to give insight into both the difficulties and chances of the sparse grid technique and to present an overview over an important facet of the wide field of hierarchical methods for visualization purposes.

2. Mathematical Foundations

In this section a brief summary of the basic ideas of sparse grids is given. For a detailed survey of sparse grids we refer to [2, 31]. In order to make this overview easier to understand and to reduce the number of indices, we describe only three-dimensional grids, whereas the sketches always reveal the two-dimensional situation.

2.1. Sparse Grids

Let $f : [0,1]^3 \longrightarrow \mathbf{R}$ be a smooth function defined on the unit cube in $\mathbf{R}^3$ with values in $\mathbf{R}$. Furthermore, f should vanish on the boundary of the cube. This condition is

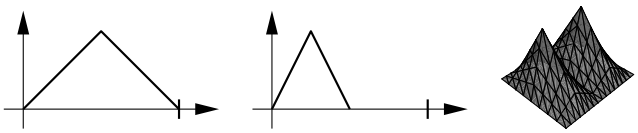


Figure 1. Examples of basis functions, $b_1^{(1)}$ and $b_1^{(2)}$ on the left and $b_{1,1}^{(1,2)}$ and $b_{1,2}^{(1,2)}$ on the right hand side.

not a strong restriction but it is just helpful for an elegant description. Of course, the algorithms can handle three-dimensional functions and even vector fields without zero boundary conditions. If such a function f is used in a numerical computation it has to be discretized, which means that only function values at certain positions of a spatial grid are stored. The simplest of these grid structures is a uniform mesh, which can be represented as a three-dimensional array.

Now let G_{i_1,i_2,i_3} be a uniform grid with respective mesh widths $h_{i_j} = 2^{-i_j}$, $j = 1,2,3$ and $\hat{L}_n$ be the function space of the piecewise tri-linear functions defined on $G_{n,n,n}$ and vanishing on the boundary. Additionally, consider the subspaces S_{i_1,i_2,i_3} of $\hat{L}_n$ with $1 \le i_j \le n$, $j = 1,2,3$, which consist of the piecewise tri-linear functions defined on G_{i_1,i_2,i_3} and vanishing on the grid points of all coarser grids, with

$$\hat{L}_n = \bigoplus_{i_1=1}^{n} \bigoplus_{i_2=1}^{n} \bigoplus_{i_3=1}^{n} S_{i_1,i_2,i_3} \quad . \tag{1}$$

Hence, we have found a hierarchical basis decomposition of the function space $\hat{L}_n$ where piecewise tri-linear finite elements are used as basis functions in each subspace S_{i_1,i_2,i_3}. These basis functions are defined as follows (compare Fig. 1):

$$b_{k_{1\cdots3}}^{(i_{1\cdots3})}(x_{1\cdots3}) := \prod_{j=1}^{3} w_{i_j}\left(x_j - (2k_j - 1)\cdot h_{i_j}\right)$$

$$\text{with} \quad w_i(x) := \begin{cases} \frac{h_i + x}{h_i} & : \quad -h_i \le x \le 0 \\ \frac{h_i - x}{h_i} & : \quad 0 \le x \le h_i \\ 0 & : \quad \text{else} \end{cases} .$$

Since we are interested in estimations of the interpolation error, we look at $\hat{f}_n \in \hat{L}_n$, the interpolated function on the grid $G_{n,\cdots,n}$, which is given by

$$\hat{f}_n = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \sum_{i_3=1}^{n} f_{i_1,i_2,i_3} \quad , \quad \text{where} \tag{2}$$

$$f_{i_1,i_2,i_3} = \sum_{k_1=1}^{2^{i_1}-1} \sum_{k_2=1}^{2^{i_2}-1} \sum_{k_3=1}^{2^{i_3}-1} c_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \cdot b_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \quad . \tag{3}$$

The values $c_{k_1,k_2,k_3}^{(i_1,i_2,i_3)}$ are called contribution coefficients and $f_{i_1,i_2,i_3} \in S_{i_1,i_2,i_3}$ is a linear combination of the basis functions of the appropriate subspace. It can be shown that

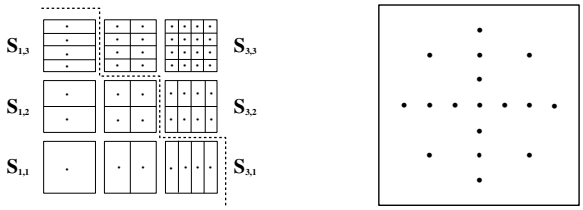


Figure 2. A two-dimensional hierarchical subspace decomposition is shown on the left hand side, the respective sparse grid is sketched on the right hand side.

the following estimations hold with regard to the L^2 or L^∞ norms (compare [2, pp. 13]):

$$\|f_{i_1,i_2,i_3}\| \quad \le \quad C \cdot \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\| \cdot h_{i_1}^2 h_{i_2}^2 h_{i_3}^2 \ , \tag{4}$$

$$\left\| f - \hat{f}_n \right\| \quad \le \quad O\left(h_n^2\right) \ . \tag{5}$$

So far we have just dealt with regular uniform meshes, which are named full grids. Now let us turn to sparse grids. Equation (4) shows that $\|f_{i_1,i_2,i_3}\|$ has a contribution of the same order of magnitude, namely $O(2^{-2\cdot const})$ for all subspaces with $i_1 + i_2 + i_3 = const$. Additionally, these subspaces have the same number of basis functions, namely $2^{const-3}$. Since the number of basis functions is equivalent to the number of stored grid points and because of the contribution argument as well, it seems to be a good idea to define a sparse grid space $\tilde{L}_n$ as follows (compare also Fig. 2):

$$\tilde{L}_n := \bigoplus_{i_1+i_2+i_3 \le n+2} S_{i_1,i_2,i_3} . \tag{6}$$

Now the interpolated function $\tilde{f}_n \in \tilde{L}_n$ is given by

$$\tilde{f}_n = \sum_{i_1+i_2+i_3 \le n+2} f_{i_1,i_2,i_3} \tag{7}$$

and the interpolation error with regard to the L^2 or L^∞ norm is given by (compare [2, pp. 23])

$$\left\| f - \tilde{f}_n \right\| \le O\left(h_n^2 \left(\log_2\left(h_n^{-1}\right)\right)^2 \right) . \tag{8}$$

This estimation shows that the sparse grid interpolated function $\tilde{f}_n$ is nearly as good as the full grid interpolated function $\hat{f}_n$ (compare Eq. (5) with Eq. (8)).

Now we consider the dimensions of the function spaces $\hat{L}_n$ and $\tilde{L}_n$, which correspond to the number of nodes of the underlying grids. Obviously, the dimension of the full grid space is given by

$$\dim(\hat{L}_n) = O\left(2^{3n}\right) = O\left(h_n^{-3}\right) . \tag{9}$$

For the sparse grid the following relation holds:

$$\dim(\tilde{L}_n) = O\left(2^n n^2\right) = O\left(h_n^{-1} \left(\log_2\left(h_n^{-1}\right)\right)^2 \right) . \tag{10}$$

level	6	7	8	9	10	11
full grid size	65^3	129^3	257^3	513^3	1025^3	2049^3
full grid	1 MB	8 MB	64 MB	512 MB	4 GB	32 GB
sparse grid	15 kB	35 kB	83 kB	200 kB	450 kB	1 MB

Table 1. Memory consumption of sparse grids

Therefore, a tremendous amount of memory can be saved if sparse grids are used instead of full grids. Table 1 demonstrates this benefit listing the memory consumption for various grid levels on the assumption that scalar single precision floating point values are given at each grid node. Obviously, sparse grids are also very suitable for compressing huge regular data sets. This opens up the potential to visualize them even on workstations with a limited amount of main memory.

Finally, recalling that the sparse grid space $\tilde{L}_n$ is the direct sum of all subspaces $S_{i,j,k}$ with $i+j+k \leq n+2$, we define the *level of a subspace* as the number $n = i+j+k-2$ and the *level of the sparse grid space* as the direct sum of all subspaces of the same level of subspaces. Hence, $\tilde{L}_n$ is the direct sum of its first n levels and is called a *sparse grid of level n*.

2.2. Adaptive Evaluation of Sparse Grids

In order to improve on the rather time consuming standard sparse grid interpolation as described above, an adaptive approach for the function evaluation is presented in this subsection.

First of all, it is important to distinguish between actual adaptive sparse grids and an adaptive way of function evaluation, i.e. an adaptive traversal of ordinary sparse grids. Adaptive sparse grids were introduced to the field of numerical simulations by Bungartz [2] in 1992. Roughly spoken, the idea of adaptive sparse grids is to store contribution values only if their norm is greater than a given error criterion. The resulting memory savings are invested in calculating a further level of the numerical sparse grid simulation. In [2] it was shown that adaptive sparse grids lead to slightly better numerical results than plain sparse grids.

However, in our situation an upper bound of the accuracy is given by the input data. Thus, we do not need adaptive sparse grids to improve accuracy. On the other hand, using sparse grids instead of full grids results in such a great advantage of memory saving that the benefit of employing adaptive sparse grids instead of plain sparse grids is negligible.

Since our goal is to decrease the computing time of the sparse grid interpolation, we introduce an adaptive traversal of the standard sparse grid in order to compute function values. Again the idea is to omit contribution coefficients with a norm below a given error criterion during the interpolation process. Here, we have to distinguish between adaptivity with regard to the L^2 and the L^∞ norm. Although they generate the same sparse grid, these norms lead to slightly different adaptive approaches.

Analyzing the situation with respect to the L^∞ norm, we find that the contribution of one basis element of subspace S_{i_1,i_2,i_3} to the function value is given by (compare Eq. (3))

$$\left\| c_{k_{1\ldots3}}^{(i_{1\ldots3})}\, b_{k_{1\ldots3}}^{(i_{1\ldots3})} \right\|_\infty = \left| c_{k_{1\ldots3}}^{(i_{1\ldots3})} \right| \cdot \left\| b_{k_{1\ldots3}}^{(i_{1\ldots3})} \right\|_\infty = \left| c_{k_{1\ldots3}}^{(i_{1\ldots3})} \right|$$

and the maximum contribution of subspace S_{i_1,i_2,i_3} is

$$\max_{k_{1\ldots3}} \left| c_{k_{1\ldots3}}^{(i_{1\ldots3})} \right| \quad \text{with} \quad 1 \leq k_j \leq 2^{i_j-1} \quad . \tag{11}$$

For vector fields the absolute value $|\cdot|$ has to be replaced by an appropriate norm of the Euclidean space $\mathbf{R}^m$ and we apply the maximum norm of $\mathbf{R}^m$ in order to ensure maximum accuracy for all components of the vector field.

The actual concept of the adaptive grid traversal is that basis functions, which have contribution coefficients with an absolute value below a given error bound, are left out during the interpolation process. This results in a function evaluation that considers the local structure of the data set.

As a second modification of the plain sparse grid algorithm, we have integrated a preprocessing step, which computes and stores the maximum contribution of each subspace (see Eq. (11)). This kind of adaptive grid traversal leads to a function evaluation with direction dependent accuracy, because different subspaces of the same level exhibit different resolutions in the three coordinate directions (reconsider the hierarchical subspace decomposition on the left hand side of Fig. 2).

Now let us discuss the adaptive approach based on the L^2 norm. A straightforward calculation shows that the contribution of one basis element of subspace S_{i_1,i_2,i_3} to the function value is given by

$$\begin{aligned} \left\| c_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \cdot b_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \right\|_2 &= \left| c_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \right| \cdot \left\| b_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \right\|_2 \\ &= \left| c_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \right| \cdot \sqrt{\left(3 \cdot 2^{i_1+i_2+i_3-1}\right)^{-3}} \, . \end{aligned}$$

Since $i_1 + i_2 + i_3 - 1 = n + 1$ with n denoting the current level, the square-root term only depends on the level and is, therefore, constant for all subspaces of the same level. Hence, this term is also a factor of the maximum contribution of the corresponding subspaces.

In contrast to the L^∞ norm, the L^2 norm generates an adaption strategy that considers not only the absolute value but also the level of a contribution coefficient. Contribution coefficients of higher levels are more likely to be omitted than coefficients of lower levels.

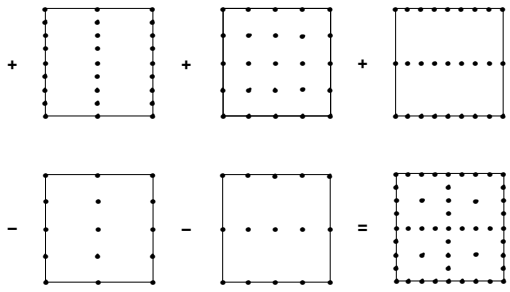


Figure 3. A two-dimensional sparse grid of level 3 can be reconstructed by linear combination of five full grids of low resolution.

2.3. The Combination Technique

Since both the standard and the adaptive sparse grid interpolation of function values are quite complicated and rather time consuming, we have also implemented the so-called combination technique, which was introduced by Griebel, Schneider, and Zenger in 1992 [11]. Actually, the combination method has been used in numerical simulations in order to combine partial solutions computed on smaller, suitable full grids to the desired sparse grid solution. However, we start with a data set given on a sparse grid and decompose the grid such that the data set is represented on certain uniform full grids of low resolution. Now the fast and simple tri-linear interpolation can be performed on each of these full grids. The resulting value is computed by linear combination of the tri-linear interpolated full grid results.

Specifically, it can be proven that the three-dimensional interpolated function $\tilde{f}_n \in \tilde{L}_n$ is given by

$$\tilde{f}_n = \sum_{i_1+i_2+i_3=n+2} f^c_{i_1,i_2,i_3} - 2 \cdot \sum_{i_1+i_2+i_3=n+1} f^c_{i_1,i_2,i_3} + \sum_{i_1+i_2+i_3=n} f^c_{i_1,i_2,i_3} \tag{12}$$

where $f^c_{i_1,i_2,i_3}$ denotes the tri-linear interpolation of function values on the respective full grid. Figure 3 reveals the two-dimensional situation, which also shows that the used full grids consist of the same nodes as the corresponding sparse grid.

Investigating the benefits of the combination technique, we find that the total number of summands of the standard sparse grid interpolation on a three-dimensional sparse grid of level n is given by

$$\sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{1}{6} n(n+1)(n+2) \tag{13}$$

(compare Eq. (7)), whereas the total number of tri-linear interpolations of the combination method adds up to

$$\sum_{i=n-2}^{n} \frac{i(i+1)}{2} = \frac{3}{2} n(n-1) + 1 \tag{14}$$

in the three-dimensional case (see Eq. (12)). That is, the number of tri-linear interpolations of the combination method is one order of magnitude lower than the number of summands of the standard interpolation.

However, the combination method outperforms the standard method in terms of basic arithmetical operations only for levels above 50, for which computers will not have enough main memory presumably for the next few decades. Despite this, the main advantage of the combination technique is the fact that uniform full grids are used. Thus, it is possible to implement the interpolation routine in terms of tight `for`-loops, which makes the combination technique an order of magnitude faster than the standard approach even for the lower levels. Additionally, it is possible to exploit the texture hardware support of graphics workstations for the interpolation of function values (see Sect. 4.2).

Finally, we have to comment on memory consumption of the combination method. Since here some points of the actual sparse grid are stored several times, the memory consumption of the combination method is about four to five times higher than the one of the standard sparse grid method. However, compared to full grids the required storage is still negligible.

3. Particle Tracing

Flow visualization tools based upon particle methods continue to be an important topic of research. Lagrange visualization techniques of vector fields are based upon the numerical solution of an initial value problem for the following ordinary differential equation:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x},t) \quad , \quad \mathbf{x}(t_0) = \mathbf{x}_0 \quad , \tag{15}$$

where $\mathbf{v}$ denotes the velocity vector field, $\mathbf{x}$ the position, t the time variable and $\mathbf{x}_0$ the start value at the initial time t_0.

Usually, a numerical integration method is used to obtain a solution. All such methods have in common that they have to evaluate the vector field $\mathbf{v}$ at certain positions, which are in general not at grid points. Therefore, the value of $\mathbf{v}$ at such a position has to be interpolated. As mentioned in Subsect. 2.1, the interpolation on sparse grids is different from the one on full grids, whereas most other parts of the particle tracing algorithm can remain unchanged. Further exceptions are the routines required for handling curvilinear grids (see Subsect. 3.4).

Our particle tracing module implements the same features, e.g. colored streak lines, ribbons, tubes, balls, and

tetrahedra (see Figs. 5 to 7), as our previous full grid particle tracing tool, which is partially described in [12] and [13]. For comparison, we have implemented several interpolation techniques using sparse grids, which are described below.

3.1. Uniform Sparse Grids

In contrast to tri-linear full grid interpolation, sparse grid interpolation does not operate locally, because one basis function in each subspace contributes to the function value. Since the tri-linear interpolation is one of the most time consuming operations during the particle tracing process on full grids [15], the complicated sparse grid interpolation is even more time consuming. Therefore, it is important to perform the interpolation as fast as possible.

The contribution coefficients of the sparse grid are usually stored in a binary tree [2, 3, 14]. In this case, a rather slow recursive tree traversal is necessary for the interpolation of function values. Although caching strategies can increase the efficiency of the traversal [14], the computation remains rather time consuming. In order to avoid the tree traversal and to accelerate the access to the contribution coefficients, we have developed a new, very efficient data structure based upon arrays, which can be accessed directly.

These data structures and the associated access methods are implemented by means of a particular C++ class hierarchy. The base class contains a stack of n levels. Furthermore, each level comprises the respective number of subspaces ($(n+1)n/2$). The subspaces themself contain an array of the size 2^{n-1} times the data dimension each, holding the contribution coefficients.

The function value at an arbitrary position is computed according to Eq. (7) by invoking an interpolation method. This method sends a method call to each level to accumulate the contributions to the resulting value. Here it performs a loop over all subspaces of the current level. In this loop, the required basis function is determined from the coordinates of the working position. Recalling that only one basis function per subspace is unequal to zero at a certain position because all basis functions are hat-functions with disjunct supports, we can easily determine the required contribution value. Now the 'height' over the current position in the tri-linear hat-function is computed and multiplied by the contribution value. Thus, we obtain the total contribution of this subspace to the function value. Additionally, we compute the Jacobian in this loop by looking up the correct 'height' of the derivative of the hat-function, which is a simple box-function. The Jacobian is needed to determine the local rotation of the flow for displaying stream bands and stream tetrahedra.

3.2. Adaptive Grid Traversal

In order to perform an adaptive grid traversal as described in Subsect. 2.2, the loop had to be enhanced in such a way that contribution values smaller than a given error bound are omitted in the function value interpolation process. In addition, the initialization method had to be modified, which is called in the preprocessing step when the actual sparse grid is created. During this process the contribution coefficients are computed from an analytic function or a full grid data set, or they are read directly from a sparse grid data set. Since we often deal with vector fields, each basis function does not only contain a single contribution coefficient but an array of coefficients. For the purpose of adaptive function evaluation, the mentioned array has been extended by one component in order to store the maximum absolute value of the contribution coefficients. Moreover, a variable has been added to each subspace for storing the maximum contribution coefficient of the entire subspace. Of course, all these additional variables storing maximum contribution coefficients are initialized during the creation of the sparse grid. Keeping in mind that the maximum contribution coefficients are the ones according to the L^∞ norm, the error criterion has to be modified, if the L^2 norm is used (compare Subsect. 2.2).

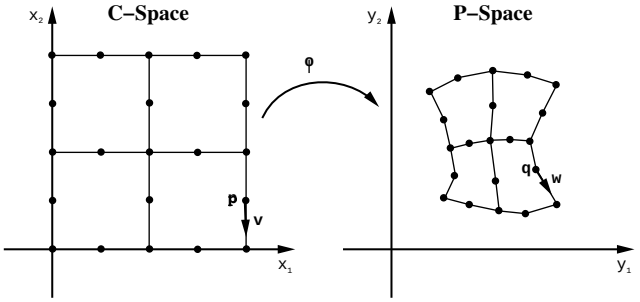
3.3. Combination Technique

Equation (12) determines the interpolation process for the combination technique, which combines full grids of low resolution to a resulting sparse grid. Since these full grids are uniform grids, the function values can be stored in three-dimensional arrays and derived by tri-linear interpolation. Thus, the interpolation method of the appropriately derived class can address the necessary function values in a tight loop. This fact makes the combination technique an order of magnitude faster than the previously described sparse grid interpolation even for low levels.

The combination technique can be accelerated by using the texture hardware of modern graphics subsystems. However, this method can only be used efficiently for volume visualization and not for particle tracing. The reason is, that for particle tracing only a single function value has to be interpolated at a given time step, whereas an entire plane of values is required for volume rendering. Subsection 4.2 gives a detailed description of employing texture hardware.

3.4. Curvilinear Sparse Grids

The underlying concept of curvilinear sparse grids is the same as for curvilinear full grids. In the case of uniform full grids, only the function values are stored in an array, whereas in case of curvilinear grids, the function values and



$$\begin{array}{lrcl} \text{Points in P-space} & \mathbf{q} & = & \varphi(\mathbf{p}) \\ \text{Vectors in P-space} & \mathbf{w}(\mathbf{q}) & = & \mathrm{Jac}(\varphi)|_{\mathbf{p}} \cdot \mathbf{v}(\mathbf{p}) \ . \end{array}$$

Figure 4. Relationship between computational and physical space.

the coordinates of the grid points as well are saved. If a curvilinear sparse grid is considered, the contribution coefficients of the coordinates of the grid points are stored as additional components of the basis functions. For the combination technique, the coordinates of the grid points are stored in the arrays of the small full grids accordingly.

The coordinates given on grid points of the sparse grid can be interpreted as a discrete version of the coordinate function φ, which relates points of the computational space (C-space) to points in the physical space (P-space) (see Fig. 4).

Particle tracing in arbitrary non-uniform grids requires the so-called *point location* to be performed for each integration step, in order to find the cell containing the actual particle position. Only then the velocity at that position can be accurately interpolated from the values at the cell vertices.

For the case of curvilinear grids, particle tracing algorithms can be divided into P-space and C-space methods. A C-space algorithm calculates the particle path in computational space where point location is as simple as for uniform grids. However, tri-interpolation in C-space requires the P-velocities at the vertices to be transformed into C-space by means of the Jacobian and the back-transformation of the path positions into P-space for visualization. In contrast, a P-space method computes the particle trace directly in the physical domain, which saves the transformations but leads to a more complex point location and interpolation. Sadarjoen et al. [21] showed that P-space algorithms are in general preferable to C-space methods. Hence, we have implemented a P-space algorithm appropriately adapted for sparse grids. Working directly in P-space, the *stencil walk* algorithm introduced by Buning [4] is usually used for the point location.

First of all, we initialize the desired C-space position $\mathbf{r}_c$ by starting in the center of our volume in C-space. In order to improve this guess, the C-space position is transformed into P-space. This is done by a sparse grid interpolation using either plain sparse grids, adaptive sparse grids, or the combination technique. If the difference of the transformed guess and the current position in P-space is small enough, we accept the C-space position. Otherwise the difference is transformed back into C-space via the inverse Jacobian and then added to the previous guess. Thereafter, the procedure is iterated until the appropriate position in C-space is located. On full grids, the stencil walk algorithm usually needs less than five iterations to find the correct C-space position.

As yet, the modifications of the stencil walk algorithm seem to be very moderate. But the main question is how to calculate the inverse Jacobian. On full grids, this is done on the fly by tri-linear interpolation of the eight Jacobians at the vertices of the current cell and subsequent matrix inversion. The Jacobians at the vertices are computed by finite differences. However, tri-linear interpolation is not possible on sparse grids. Thus, we have to use sparse grid interpolation and we have to store the inverse Jacobian, i.e. the respective contribution coefficients, at each sparse grid point. This memory overhead can only be justified with the fact that sparse grids themselves are very storage efficient.

As the data sets usually do not contain the Jacobians explicitly, the Jacobians and their inverse matrices have to be calculated as well as their contribution coefficients during initialization. We have modified the methods in such a way that in a first pass the contribution coefficients of the function values and the inverse Jacobians are computed and stored in the sparse grid structure. In a second pass, the contribution coefficients of the inverse Jacobians are computed and stored over the original components of the inverse Jacobians. The second pass traverses the levels beginning with the highest level and ending with level 1, because the contribution coefficients only depend on the current function value and on the function values of lower levels. This fact can be deduced from Eqs. (2) and (3). Hence, it is possible to overwrite the original components of the inverse Jacobians successively. Notice that level 0 is not part of the mentioned second pass because in level 0 contribution values and function values coincide.

3.5. Examples and Results

In order to compare our sparse grid particle tracing module with full grid particle tracers, two data sets have been used. The first one is a cavity flow data set (see Fig. 5) on a full grid of 129^3 nodes, which corresponds to level 7 in sparse grid terminology. The data set contains velocity, pressure, and temperature data at each vertex requiring more than 40 MB. The same data set with a resolution of 257^3 (level 8) would need more than 320 MB, which is probably too much for most workstations. On the other

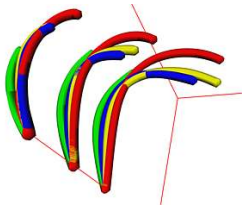


Figure 5. Streak tubes in a cavity flow computed on a full grid and sparse grids of level 3, 5, and 7.

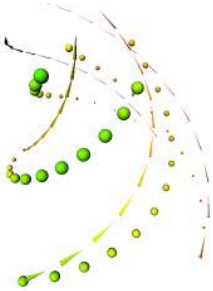


Figure 6. Colored streak balls and tetrahedra in a vortex flow.

Figure 7. Streak bands in a vortex flow computed on a full grid and sparse grids of level 0 (left), 1 (middle), and 4 (right).

hand, this data set stored on a sparse grid consumes only 175 kB for level 7 and 415 kB for level 8.

The second data set is a vortex flow (compare Figs. 6 and 7) given analytically. Therefore, we are able to create sparse and full grids in any resolution only limited by the main memory of the used machine.

For each experiment nine streak ribbons consisting of about 500 particles were integrated using an adaptive RK3(2) scheme (see Fig. 7). Time measurements have shown that interactive particle tracing is possible even on sparse grids of level 8. However, the drawback of sparse grid interpolation is that the computing time rises at least quadratically if the grid level is increased. In contrast to this, the computing time of the full grid module is only growing slowly, since in theory the time for particle tracing on full grids is independent of the grid size. Investigating the accuracy of sparse grid particle tracing, theory (see Eqs.(5) and (8)) tells us that the difference should be rather small. Moreover, the integration error of RK3(2) is on the order of $O(\tau^3)$ where τ denotes the current time step (see the discussion in [25]). In fact, the results of particle tracing on the analytic data set confirm these estimations, since the ribbons computed on full and sparse grids coincide on screen for levels greater than 3 (compare Fig. 7). However, for the derivation of the mentioned upper bounds for the interpolation errors, a certain smoothness of the data was a prerequisite. Since discrete data sets are not smooth

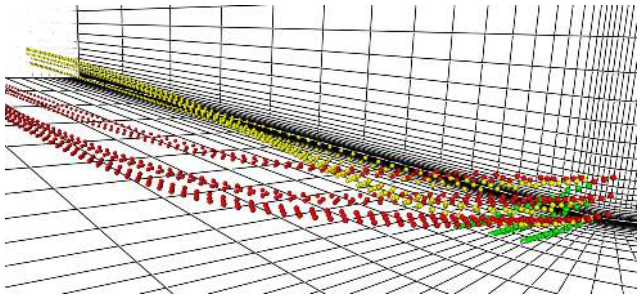


Figure 8. Streak balls in the blunt fin data set, computed on curvilinear sparse grids of level 2, 3 and 4.

at all, these estimations do not hold in this case. Indeed, for discrete data Fig. 5 reveals that the particle traces computed on sparse grids converge rather slowly to the full grid solution. Nevertheless, due to the great advantage of low memory consumption, it is possible to use a sparse grid of a sufficiently high level to overcome this problem.

For the adaptive grid traversal, several experiments have shown that it is important whether the L^2 norm is used for the adaptive traversal or the L^∞ norm. Employing the L^∞ norm leads to a marginal decrease in computing time but to a significant loss in accuracy. However, by using the L^2 norm, it is possible to decrease computing time by about 20 per cent and to achieve nearly the quality of the corresponding plain sparse grid.

The next approach for accelerating particle tracing on sparse grids is the combination technique. The first advantage of this technique compared to adaptive grid traversal is the fact that there is no loss in accuracy at all. Combination technique and plain sparse grid interpolation create exactly the same particle path. The second and more important benefit is that the combination technique is almost by a factor of four faster than plain sparse grid interpolation.

Now let us turn to curvilinear sparse grids. For a first test we have used the well-known blunt fin data set (see Fig. 8). Additionally, our module has been verified with several analytic data sets. On the one hand side these tests have confirmed that smooth data sets are more appropriate for using sparse grid methods than discontinuous data. On the other hand these tests have revealed that particle traces calculated on curvilinear sparse grids converge slower to the corresponding full grid trace. The reason for this decline in accuracy might be due to a less accurate point location caused by an intensive use of sparse grid interpolation in the stencil walk algorithm. Finally, time measurements have shown that particle tracing on curvilinear sparse grids is about five times slower than tracing on uniform grids. This is roughly the same deceleration as on full grids. Nevertheless, interactive particle tracing is possible on curvilinear sparse grids of level 7 by using the combination technique.

4. Volume Rendering

Three-dimensional scalar data sets resulting from measurement or numerical simulation can be visualized very well by volume rendering techniques. Direct volume rendering tries to convey a visual impression of the complete data set by assigning different color and opacity values to different objects or value ranges within the volume. The resulting image is then computed by taking into account the so defined emission and absorption effects as seen by an outside viewer. The underlying theory of the physics of light transport is simplified to the well known volume rendering integral in the case of neglecting scattering and frequency effects [16]. Given the emission q and the absorption κ the intensity I along the ray s can be computed from:

$$I(s) = \int_{s_0}^{s} q(s') e^{-\int_{s}^{s'} \kappa(s'')\, ds''}\, ds'$$

The discretization of this integral together with the assumption that the mapping from scalars to RGBA values can be described by transfer functions results in the compositing formulas for computing the intensity contribution along one ray of sight:

$$I = \sum_{k=1}^{n} C_k \prod_{i=0}^{k-1} (1 - \alpha_i)$$

The emission of the voxel C_k and its opacity α_k are derived from the transfer functions after interpolation of the scalar value from the discrete sample points.

The basic ray tracing idea [18] is to shoot a ray of sight through every pixel into the volume, reconstructing the function value at appropriately chosen sample points along the ray and blending the mapped color and opacity values. Various variants of this technique exist. X-ray like images are generated by neglecting the opacity and just summing up all values. The so-called maximum intensity projection method (MIP) determines the intensity of a pixel to be the maximum function value occurring along the corresponding ray. Even iso-surfaces can be rendered if an illuminated pixel is displayed if the difference of the current function value and the iso-value changes sign. In this case, the surface normal needed for the lighting computation coincides with the direction of the function gradient.

Acceleration of this expensive technique is achieved by adaptive sampling [9, 7], by exploiting coherence [17], by parallelizing in image and object space, and by exploiting hardware in graphics workstations [5] or in special purpose architectures [20]. Related to our work are approaches where volume rendering is performed on compressed data sets. Wavelet based techniques [28, 19] reconstruct the function value from a wavelet decomposition instead from tri-linear interpolation, leaving the basic volume rendering algorithm unmodified.

4.1. Software Based Interpolation

So far the only possibility to visualize data given on a sparse grid was to expand the sparse grid to a full grid and then to use the traditional techniques on the full grid. For larger sparse grids this approach is prohibitive, because the full grid does no longer fit into the main memory of standard workstations. Additionally, the process of expanding a sparse grid is extremely slow.

Therefore, we follow the idea of other compression domain volume rendering techniques and use a traditional ray casting algorithm with the tri-linear interpolation substituted by sparse grid interpolation. Of course we employ the different interpolation techniques presented in the Sect. 3. Specifically, we reuse the methods encapsulated in the C++ classes for the standard method and the combination technique. Currently, curvilinear grids are not supported, though. These grids would require a stencil walk inside the innermost loop of the ray caster, which would slow down the interpolation process significantly.

Due to the nature of volume visualization, a huge number of sampling points is needed in the ray casting process. Even with the combination technique, rendering times are far away from interactivity. However, in contrast to particle tracing, the volume rendering process addresses sampling points in equidistant planar slices of the volume. This order can be utilized to accelerate the interpolation process by using graphics hardware, which is described in the following section. We are going to see that hardware based interpolation can only be performed in fix point arithmetic. Therefore, it is not as accurate as the software based interpolation, which can still be reasonable for generating high quality images.

4.2. Texture Hardware Based Interpolation

When interpolating data using the combination technique the processor spends most of its time on the tri-linear interpolation of the full grids. In order to significantly reduce the computation time, we decided to take advantage of the texturing hardware of modern graphics workstations. The graphics engines of these workstations have hardware based support for MIP mapped texturing. This technique reduces texture mapping artifacts during minification if pixels are covered by many texture elements. In this case texture filtering with large kernel sizes would be appropriate which cannot be done in real time. Therefore, several downsampled versions of the texture are stored together with the original resolution. Tri-linear interpolation in hardware is used to interpolate between texture levels and within the texture.

As a variant of this technique several vendors have implemented real tri-linear interpolation for 3D textures map-

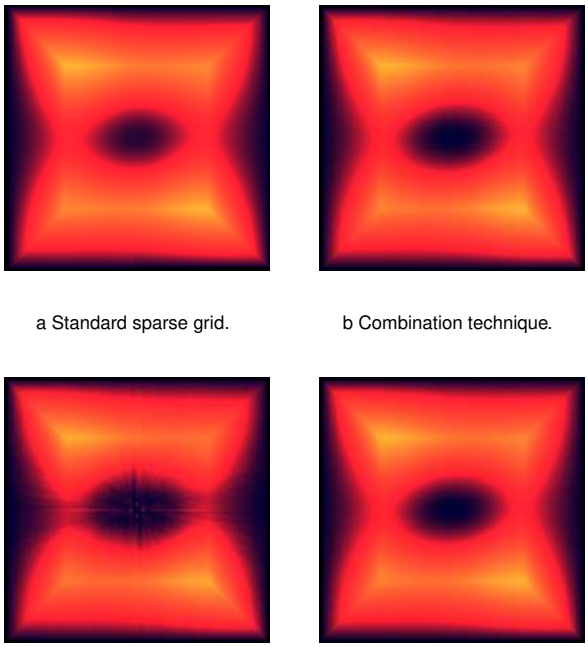
ping [22]. While originally intended to be used for volumetric effects like fog or wood texture, 3D textures turned out to be a key feature for interactive volume rendering of regular grids [30]. The basic functionality provided by the underlying OpenGL extension can be described as follows. Given a flat polygon whose arbitrary position and orientation within the volume texture is defined by appropriate 3D texture coordinates assigned to each vertex, a 2D texture is reconstructed on the polygon by tri-linear interpolation.

In our case of the combination technique for sparse grids we do not use the graphics hardware for the actual volume rendering, that means for viewing and compositing, but we employ 3D textures, tri-linear interpolation and blending operations in order to implement Eq. (12). To exploit the provided functionality we had to rewrite our algorithms in such a way that they simultaneously reconstruct all values in an entire plane, which of course is oriented perpendicular to the viewing direction of the volume rendering. Our ray caster then proceeds by compositing along rays through all textured image slices.

In detail, after defining 3D textures with the values of the full grids $f^c_{i_1,i_2,i_3}$, our class draws the appropriate plane automatically performing the tri-linear reconstruction. Then the partial results have to be added and subtracted according to Eq. (12). This is implemented by a blending step using blending extensions of OpenGL.

In order to switch quickly between the different volume textures, they all have to fit into texture memory at the same time. By using texture objects of OpenGL, preloaded textures can be selected for rendering almost instantly. The combination technique uses full grids of size $(2^i+1)\times(2^j+1)\times(2^k+1)$. On the other hand, volume textures have to be of size $2^p\times 2^q\times 2^r$. In order to fit the full grids completely into volume textures, we have to allocate textures of size $2^{i+1}\times 2^{j+1}\times 2^{k+1}$. Hence, quite a lot of texture memory is wasted, although it is a scarce resource. In principle, OpenGL supports volume textures featuring a so called border of size 2 in every direction such that the mentioned full grids would fit almost seamlessly. Unfortunately, these texture borders are not implemented in todays hardware.

When using graphics hardware for mathematical computations, accuracy can be quite a problem. On Silicon Graphics machines the blending operation can currently be performed in the frame buffer with 16 bits at most. Since pixel values are automatically clamped to values in the interval $[0,1]$, all texture elements have to be scaled down by the number of functions contributing positive values to Eq. (12). For a level 10 sparse grid, 91 of 136 functions contribute positive data, which means a loss of almost 7 bits, resulting in only 9 bits of accurate information. Since we have at best 16 bits of accurate information in the frame buffer, it is sufficient to use only 2 bytes of texture memory per voxel. If future hardware will incorporate larger frame buffers with higher pixel accuracy, this will automatically enhance the image quality of our algorithm. Although visible artifacts are remarkably small, some can be seen in Fig. 9c for an example with an effective accuracy of 7 bits.



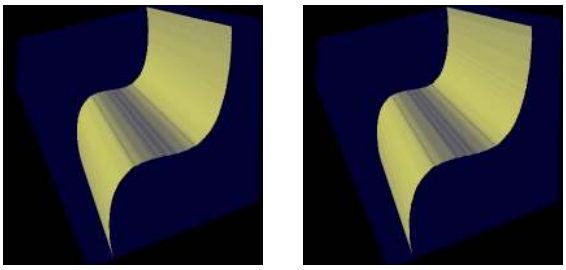
a Standard sparse grid. b Combination technique. c Plain OpenGL. d Accumulation buffer.

Figure 9. Views of the cavitiy flow rendered with different interpolation methods.

By using the accumulation buffer, these artifacts can be reduced, so that they are barely visible (see Fig. 9d). Because the frame buffer has to be combined with the accumulation buffer for every plane drawn, its usability strongly depends on graphics pipes that provide hardware based accumulation buffers like SGI's Infinite Reality system. Additionally, off-screen rendering using SGI's *P-buffer* extension is currently not possible with this approach.

4.3. Examples and Results

We tested our implementation with several data sets. Two of them are cavity flow data sets, given on a full grid of level 6, i.e. 65^3 nodes. These data sets are the result of a numerical flow simulation and contain pressure and temperature distributions of the flow. The third data set, given on a full grid of level 8 (257^3 nodes), contains a spherical harmonic (Legendre's function), which displays a solution of the Schrödinger equation of a hydrogen atom. In addition, we used an analytic test function and a discontinuous one for considering interpolation quality.



a Combination technique. b Full grid.

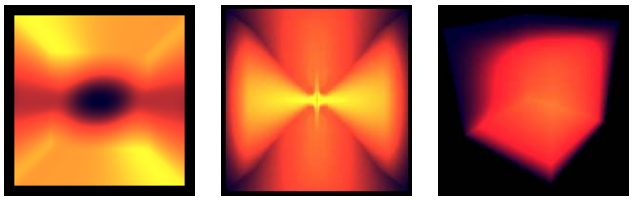
Figure 10. Iso-surface of temperature in a cavity flow.

Figures 9a to 9d show X-ray images of the pressure values in the mentioned cavity flow. These images have been rendered in order to reveal differences between the four implemented interpolation algorithms. In the OpenGL image rendered without using the accumulation buffer, small flaws can be detected, whereas the other sparse grid methods render nearly identical images.

In the other figures, sparse and full grid results are compared using different lighting models and data sets. The two images in Fig. 10 display iso-surfaces of the temperature in the cavity flow data set. In Figs. 11a and 12a, the pressure of the same data set is visualized by means of the maximum intensity projection lighting model. In further maximum intensity projection images (see Figs. 11b and 12b), the data set of a spherical harmonic function is visualized. Then, the temperature distribution in the cavity flow is depicted in Figs. 11c and 12c by using an X-ray lighting model. Finally, Figs. 13a to 14c show that the smoothness of extracted iso-surfaces depends on the used grid level.

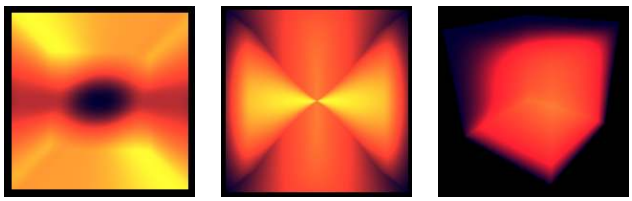
Considering the performance of our volume visualization program we find that the time consumption of the X-ray and maximum intensity projection algorithms is data independent and exactly the same, while iso-surface rendering takes about two times as long as the other methods. Moreover, the total time of the iso-surface computation depends on the size of the extracted surface.

We tested our implementation on several Silicon Graphics workstations, ranging from the O2 to an Onyx with RealityEngineII and a BaseReality pipes. Since the O2 systems do not have hardware based 3D texture support, only the software based interpolation schemes can be used in a sensible way on these machines. We realized that rendering times of both software implementations approximately increase by a factor of 1.4 every time the used level rises by one, whereas the measured times of the hardware based implementation increase by a factor of 1.2 . The speed of the interpolation on full grids does not depend on the used level, as anticipated. The combination technique is between seven and ten times faster than the sparse grid algorithm. Furthermore, the OpenGL hardware method is between 25 and 60 times faster than the software combination technique. This results in a speed up factor between 200 and 450 from the sparse grid to the hardware based method. Hence, we are able to perform volume visualization on sparse grids interactively exploiting the texture hardware for acceleration purposes.



a MIP image of pressure distribution. b MIP image of a hydrogen atom. c X-ray image of temperature distribution.

Figure 11. Images calculated on a sparse grid.



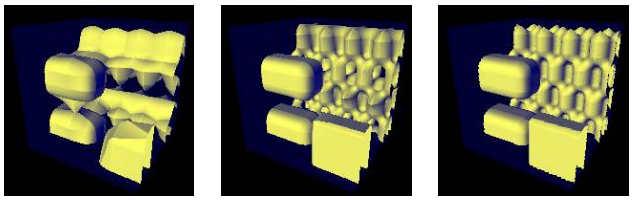
a MIP image of pressure distribution. b MIP image of a hydrogen atom. c X-ray image of temperature distribution.

Figure 12. For comparison: Images calculated on a full grid.

The combination technique requires about four to five times the memory of the actual sparse grid algorithm since some of the needed nodes are stored several times. The OpenGL version of the combination methods consumes about two and a half times the memory of the software version, because each of the used textures has to have dimensions that can be written as two to the i-th power. Nevertheless, compared with the original full grid data set, both implementations of the combination technique require a negligible amount of memory.

5. Conclusion

We have presented sparse grids as a competing approach for the compact representation of three-dimensional data sets by means of hierarchical basis functions. We have introduced two important visualization methods, particle tracing and volume ray casting, working on the sparse grid representation. They allow to carry out flow and volume visualization directly on the results from a numerical sparse



a Level 4. b Level 6. c Full grid.

Figure 13. Iso-surface of a discontinuous test data set given on a sparse grid and on a full grid for comparison.

a Level 4. b Level 6. c Full grid.

Figure 14. Iso-surface of an analytical function given on a sparse grid and on a full grid for comparison.

grid simulation without prior transformation to the associated full grids. This is an important step for the broader application of the sparse grid method, since in real applications it is often impossible to load full grids of more than 512^3 nodes into the main memory of a workstation for visualization purposes. Furthermore, the sparse grid approach can be used as a compression method in order to realize volume rendering of huge regular data sets on workstations with a small amount of main memory.

Technically, we have presented several methods to accelerate the sparse grid interpolation process. For particle tracing, we have implemented adaptive sparse grids with error monitoring and the combination technique. By using texture graphics hardware, we have been able to overcome the tardiness of sparse grid interpolation in the case of volume visualization. Therefore, it is possible to use flow and volume visualization on sparse grids interactively, in contrast to other compression approaches.

Compared to the well-known wavelet compression [6, 8], we can state advantages and disadvantages. The main benefit of the wavelet compression is the fact that the wavelet decomposition is data dependent, which means that the resulting compression is by itself adapted to the underlying data set. Comparing the decomposition process we find the wavelet decomposition being comparatively difficult, whereas the sparse grid decomposition is conceptually simple. On the other hand, the wavelet reconstruction is quite simple, whereas the sparse grid reconstruction, i.e. the interpolation, is rather complicated and very time consuming. However, the basis functions used in the case of sparse grids are so simple (compact support and piecewise tri-linear) that the texture hardware can perform the reconstruction in connection with the combination method. Hence, in the end it turns out that the hardware assisted sparse grid volume visualization is much faster than visualization methods working on other compressed data sets.

We conclude this section by describing two scenarios where the visualization process can take advantage of sparse grid methods. First, we assume that a sparse grid data set resulting from a numerical simulation is given. Then, there are two possibilities for the visualization. The traditional approach is to interpolate the data set once into a huge full grid data set (see Table 1). If the resulting full grid squeezes into the main memory of the machine, fast full grid volume visualization methods can be performed. In contrast to this, our strategy is to use the OpenGL algorithm for getting the first images of the data long before the traditional interpolation process will be finished. If the full grid data set does not fit into the main memory, a direct sparse grid visualization method has to be performed anyway. As a second scenario, we assume that a huge full grid data set should be visualized, which results from a numerical simulation or from extensive measurements. Now, the sparse grid method can be used to compress the huge data set such that it will fit into the main memory of a workstation. Then, it is possible to visualize the compressed data using the techniques presented in this paper.

6. Future Work

There are several directions of future work. Concerning volume visualization, the first goal is to implement more sophisticated transfer functions and lighting models into our visualization program, for instance an emission-absorption model. As a second goal, we intend to use OpenGL in order to accelerate the surface and volume illumination as well. This approach is already used in case of volume rendering on full grids as described in [24, 29, 30]. As far as particle tracing is concerned, additional visualization techniques could be implemented on sparse grids. Feasible directions are texture based algorithms and iconic methods combined with feature extraction. Finally, our sparse grid particle tracer could be extended to multi-block data sets in the same way as it has been done in our full grid particle tracing module [13].

7. Acknowledgments

We are grateful to S. H. Enger from the Department of Fluid Mechanics of the University of Erlangen for making

the cavity data set available to us. For helpful discussion, we would like to thank Ch. Zenger and the members of his group at the Computer Science Department of the University of Munich and K. Frank from the High Performance Computing Center Stuttgart. In particular, we gratefully acknowledge the encouragement of R. Grosso from AEA Technology during the initial phase of sparse grid visualization. Finally, we wish to thank our colleagues K. Hormann for his help with some mathematical issues and R. Westermann for his valuable remarks on OpenGL programming.

This work was partially supported by FORTWIHR, the Bavarian Consortium for High Performance Scientific Computing, and by the DFG, the German Research Foundation.

References

[1] K. I. Babenko. Approximation by Trigonometric Polynomials in a certain Class of Periodic Functions of Several Variables. *Soviet Mathematics*, 1:672–675, 1960. Translation of Doklady Akademii Nauk SSSR.

[2] H.-J. Bungartz. *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung*. PhD thesis, TU Munich, 1992.

[3] H.-J. Bungartz and T. Dornseifer. Sparse Grids: Recent Developments for Elliptic Partial Differential Equations. In *Multigrid Methods V*, Lecture Notes in Computational Science and Engineering. Springer-Verlag, 1998.

[4] P. Buning. Numerical Algorithms in CFD Post-Processing. In *Computer Graphics and Flow Visualization in Computational Fluid Dynamics*, number 1989-07 in Lecture Series, Brussels, Belgium, 1989. Von Karman Institute for Fluid Dynamics.

[5] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In A. Kaufman and W. Krüger, editors, *1994 Symposium on Volume Visualization*, pages 91–98. ACM SIGGRAPH, 1994.

[6] C. K. Chui. *An Introduction to Wavelets.* Academic Press, Inc., San Diego, 1992.

[7] J. Danskin and P. Hanrahan. Fast Algorithms for Volume Ray Tracing. *Transactions Workshop on Volume Visualization*, pages 91–98, 1992.

[8] I. Daubechies. *Ten Lectures on Wavelets.* Number 61 in CBMS-NSF Series in Applied Mathematics. SIAM, Philadelphia, 1992.

[9] B. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. *Computer Graphics (SIGGRAPH '88)*, 22(4):65–74, August 1988.

[10] M. Griebel, W. Huber, U. Rüde, and T. Störtkuhl. The combination technique for parallel sparse-grid-preconditioning or -solution of PDE's on multiprocessor machines and workstation networks. In L. Bougé, M. Cosnard, Y. Robert, and D. Trystram, editors, *Second Joint International Conference on Vector and Parallel Processing*, pages 217–228, Berlin, 1992. CONPAR/VAPP, Springer-Verlag.

[11] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *International Symposium on Iterative Methods in Linear Algebra*, pages 263–281, Amsterdam, 1992. IMACS, Elsevier.

[12] R. Grosso, M. Schulz, and T. Ertl. Fast and Accurate Visualization of Steady and Unsteady Flows. Technical Report 3, University of Erlangen, 1996.

[13] R. Grosso, M. Schulz, J. Kraheberger, and T. Ertl. Flow Visualization for Multiblock Multigrid Simulations. In P. Slavick and J. J. van Wijk, editors, *Virtual Environments and Scientific Visualization '96*, Heidelberg, 1996. Springer-Verlag. Proceedings of the Eurographics Workshop in Prague, Czech Republic.

[14] N. Heußer and M. Rumpf. Efficient Visualization of Data on Sparse Grids. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization*, pages 31–44, Heidelberg, 1998. Springer-Verlag.

[15] D. N. Kenwright and D. A. Lane. Optimization of Time-Dependent Particle Tracing Using Tetrahedral decomposition. In G. M. Nielson and D. Silver, editors, *Visualization '95*, pages 321–328, Atlanta, Georgia, 1995. IEEE Computer Society, IEEE Computer Society Press.

[16] W. Krüger. The Application of Transport Theory to the Visualization of 3D Scalar Data Fields. In A. Kaufman, editor, *Visualization '90*, pages 273–280. IEEE Computer Society, IEEE Computer Society Press, 1990.

[17] P. Lacroute. Analysis of a Parallel Volume Rendering System Based on the Shear-Warp-Factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, 1996.

[18] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[19] L. Lippert and M. H. Gross. Fast Wavelet Based Volume Rendering by Accumulation of Transparent Texture Maps. *Computers Graphics Forum (EUROGRAPHICS '95)*, 14(3):432–443, 1995.

[20] H. Pfister and A. Kaufman. Cube-4 — A Scalable Architecture for Real-Time Volume Rendering. In R. Crawfis and C. Hansen, editors, *1996 Symposium on Volume Visualization*, pages 47–54. ACM SIGGRAPH, 1996.

[21] A. Sadarjoen, T. van Walsum, A. J. S. Hin, and F. H. Post. Particle Tracing Algorithms for 3D Curvilinear Grids. In *Fifth Eurographics Workshop on Visualization in Scientific Computing*, 1994.

[22] Silicon Graphics Inc., Mountain View, California. *OpenGL on Silicon Graphics Systems*, 1996.

[23] S. A. Smolyak. Quadrature and Interpolation Formulas for Tensor Products of certain Classes of Functions. *Soviet Mathematics*, 4:240–243, 1963. Translation of Doklady Akademii Nauk SSSR.

[24] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. An Interactive Visualization and Navigation Tool for Medical Volume Data. In N. M. Thalmann and V. Skala, editors, *WSCG '98, The Sixth International Conference in Central Europe on Computer Graphics and Visualization '98*, volume II, pages 362–371, Plzen, Czech Republic, February 1998. University of West Bohemia Press.

[25] C. Teitzel, R. Grosso, and T. Ertl. Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes. In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing '97*, pages 31–41, Wien, April 1997. Springer-Verlag. Proceedings of the Eurographics Workshop in Boulogne-sur-Mer, France.

[26] C. Teitzel, R. Grosso, and T. Ertl. Particle Tracing on Sparse Grids. In D. Bartz, editor, *Visualization in Scientific Computing '98*, pages 81–90, Wien, April 1998. Springer-Verlag. Proceedings of the Eurographics Workshop in Blaubeuren, Germany.

[27] C. Teitzel, M. Hopf, and T. Ertl. Volume Visualization on Sparse Grids. *Computing and Visualization in Science*, (4):1–13, 1999.

[28] R. Westermann. A Multiresolution Framework for Volume Rendering. In A. Kaufman and W. Krüger, editors, *1994 Symposium on Volume Visualization*, pages 51–58. ACM SIGGRAPH, 1994.

[29] R. Westermann and T. Ertl. The VSBUFFER: Visibility Ordering unstructured Volume Primitives by Polygon Drawing. In R. Yagel and H. Hagen, editors, *Visualization '97*, pages 35–43, Phoenix, Arizona, 1997. IEEE Computer Society, IEEE Computer Society Press.

[30] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In M. Cohen, editor, *Computer Graphics Proceedings*, Annual Conference Series, pages 169–177, Orlando, Florida, 1998. ACM SIGGRAPH.

[31] C. Zenger. Sparse grids. In *Parallel Algorithms for Partial Differential Equations: Proceedings of the Sixth GAMM-Seminar*, Kiel, 1990.